Report No. 217 Math COO-1469-0046

17

TIME SHARING AT ILLINOIS: EXPERIENCE AND PLANS

by

C. W. Gear

December 5, 1966

DEPARTMENT OF COMPUTER SCIENCE · UNIVERSITY OF ILLINOIS · URBANA, ILLINOIS

Report No. 217

TIME SHARING AT ILLINOIS: EXPERIENCE AND PLANS

by

C. W. Gear

December 5, 1966

Department of Computer Science
University of Illinois
Urbana, Illinois 61801

1. Introduction

This report will discuss the status of the experimental time-sharing system implemented on the ILLIAC II at Urbana and attempt to pass on to the reader the lessons that have been learned as a result of this work. The second part of the report will discuss the plans for the next system. This new system will be a part of the computer system providing general service for the Urbana campus. This future system will consist of IBM 360-50/75 main processors with links to other campus computers.

This report is not intended for the expert in the time-sharing field but rather for the system man who is about to consider a large system and who would like to learn of some of the problems.

2. Present Hardware

The processing capabilities of the present system are provided by the ILLIAC II with a 52 bit, 8K 1.7 micro second main memory 65K, 7 micro second/word drum, and 10M words of disk file (two IBM 1301's at 100 micro seconds/word) plus 10 magnetic tapes. A Digital Equipment Corporation PDP-7 and 630 is connected as an I/O channel with communication facilities for up to 64 serial low-speed I/O devices (in the range 100 to 2,000 bits per second). These serial I/O lines are used to connect the main terminals which are Teletype Model 33 or 35 devices via the Bell System 103A1 data phones. The phone input is such that a single telephone number accesses all available inputs. The hardware is indicated in Figure 1.

3. Present Software

The software available is a hodge-podge of experimental programs each of which has been modified many times in order to test various ideas. The essential components of a time-sharing system are a general supervisory program, a file manipulation program, compilation and assembly programs, and ability to interact with an executing program. Perhaps the simplest form of a time-sharing system is one which provides only a minimum supervisory system as a skeleton to allow the user to interact with any program he chooses to

to load. These programs can then include the file manipulation, translation, and other service programs. It is our feeling that in general this leads to high overhead of a system program; therefore, in the Illinois System, much of the user's work is done by common programs which we then consider to be part of the general supervisory system. In our particular case the supervisory system includes a file manipulation subsection which allows the inputting of files from the console or via the off-line card reader, the outputting of files at the console or via the off-line printer-punch combination and the updating of files. The files are maintained symbolically on the disk file. The general supervisor also allows these symbolic files to be processed by the Fortran compiler or system assembler in order to obtain executeable programs. It allows programs to be loaded into core from these binary object programs. It provides the ability to start, stop, examine, or change programs in "core" and provides the ability to save a complete core load and reload it for use at a later time. The core load referred to is, of course, not kept permanently in core but switched in and out according to a scheduling algorithm. The program in core has access to system programs which communicate with the console so that direct input/output can be performed (this is, of course, buffered so that CPU time is not lost while waiting on the user's response). Finally, a simple interpreter is provided which enables low level calculations to be performed rapidly without all the messing around that would be needed if the regular system compiler were to be used.

A major lack in the present software is a conversational compiler sufficiently similar to the standard system compiler such that programs can be debugged rapidly. Such a program is under construction at this time.

4. Experience

In the process of preparing the system we have thrown far more code away than we are using. Some of this waste could have been saved by prior simulation studies, but the majority could only have been found to be less than adequate by actually using it. This section will discuss a number of isolated points concerning hardware and software that have caused us problems and in some cases have been corrected.

A. Terminals

We initially used IBM 1050 terminals experimentally but ran into a number of problems. In the first place they were unreliable in our environment and in the second place they had many features built into the logic that were undesirable for our purposes. If they had been used exactly as intended by the original designer, they would have probably been OK; but we found that a lot of code was used to overcome some of the problems introduced by logic that one would rather not have been paying for. For example, the terminal can communicate in one direction at a time only and the turnaround can only be initiated by a designated controller. This means that in order to continue to accept information from a user terminal while retaining the ability to accept output from the computer should the need arise, it is necessary to continuously cycle the controllers from a transmit to a receive status and back. It is also necessary to relay a specific end-of-message character from both the user and the computer in order to effect a turnaround.

Because of the problems, we switched to Teletype terminals. They are mechanically simple, require negligible electronics to interface to a data phone, or to interface to a DC loop. They can be operated in either the half-duplex or full-duplex so our 630 communication box was constructed for such terminals. In half-duplex both the transmit and receive portions of the terminal and the computer are permanently connected to a single communication channel. Anything that is typed at the console appears on the printer and is transmitted to the computer. This enables "echo checking" to be done. The computer reads the character that is transmitted and checks it against the original. This guards against transmission errors, and also allows the terminal user to "break" transmission at any time by typing during output by the computer. The computer can be programmed to accept multiple echo checks as a user interrupt.

In full-duplex there are two separate channels, one from the computer to the printing mechanism and one from the keyboard input to the computer. They are logically separate although both can be transmitted over one standard phone line. When the user types at a full-duplex terminal, nothing is printed unless the computer responds by transmitting the character back to the terminal. This introduces a delay of two character times (1/5 of a second) which some people find unnerving, but we think that a user would very quickly adjust. With hindsight we would have used full-duplex, since it can be programmed to do any-

thing that half-duplex can exept for echo checking and it has one further important advantage: the competent user does not require that his keyboard input be printed; he will use the computer response as a check on his typing accuracy, so he can continue to input ahead of the computer response, thus achieving overlap of his typing with the unavoidable waiting for computer service.

B. Use of Auxiliary Storage

Whenever there is more than one level of storage, there is an assignment problem. Two levels are not so bad since the main level of storage can only be used for the monitor, incoming, executing, and outgoing program plus some buffer space. Three levels poses an additional problem since the drum is an order of magnitude faster than the disk (13 times the word rate, 16 times the average access rate). We planned and we schemed over what should go where (one buffer per console on the drum, last use system processor on the drum, etc.). Unfortunately, any decision is only effective for a particular combination of user demands. The change that would be expected from, say, a high console use period in the middle of the afternoon to a high background use period rate at night will make any absolute assignment less than efficient.

Our solution--which was certainly not new--was to essentially forget that we had a drum and use the disk for everything--except that the disk programs were modified so that they used the drum as an intermediate buffer device. Unfortunately, it is not possible to transmit directly from the drum to the disk or visa versa, but by allocating a buffer area in the monitor, the transmission can be overlapped with processing. In the proposed scheme all disk writes are converted to drum writes which are an average of 16 times faster (in the case that a multiple block transfer is to be performed they are, in fact, about 20 times faster). The contents of drum are indexed by a table kept in core so that reads from the disk can be converted to reads from the drum if the information is already on the drum. If a read requires access to the disk then a copy is also kept on the drum in case it is needed again. As the drum starts to fill up, information not recently used is copied back to the disk. In order to improve the efficiency a little more, additional requests to the disk subroutine were added. These requests included the "pre-need read" which told the system that a block from the disk file would be needed later on. Its purpose

was to encourage the system to move it to the disk file so that a disk access would not cause a wait at the time the information is needed, and a "write for good" request which indicated to the system that the block concerned need not be maintained on the drum since it would not be used in the foreseeable future.

The advantages of this type of system approach to the use of multiple level auxiliary storage are that it saves program everywhere but in the disk file package, it saves making wrong decisions about the assignment of drum space (it is now done dynamically), and it is easier to reduce disk head travel by using algorithms to determine which of the blocks on drum to copy back to the disk. This sort of algorithm can be done on a machine with a large main memory but is impractical in a machine with a memory the size of ILLIAC II.

C. Telephone Lines

The use of a public switched network has its advantages and disadvantages. The main points to be made in its favor are the almost immediate availability of a line to the majority of places in which a console is to be located, and the possibility of using the switching capability of the phone system to interface more potential consoles than there are inputs to the computer. If the terminal device is sufficiently cheap, then many consoles can be placed as close as possible to potential users (in their offices if necessary). Because not all of these users will be active at one time, it is not necessary to provide inputs for every one of them.

The attendant disadvantages are the additional cost of phone lines, and of providing protection against unauthorized use. The cost of a hard-wired D.C. loop connection over a short distance (2,000 feet) is almost negligible compared with the cost of two dataphones and two phone lines (about $70/month). Because anybody with a phone has access to the computer input, the system is faced with protection against accidental and deliberate misuse. It must be possible to detect (in order):

(i) Non-data calls (from a voice phone),
(ii) Unauthorized consoles (they may not use the same code),
(iii) Unauthorized users.

It must also be possible to disconnect the caller who does not meet the

restrictions, otherwise an input to the computer will be tied up unnecessarily. A hard-wired console is known to pass the first two tests, and if an unauthorized user insists on remaining at a console although the computer refuses to acknowledge him, one can rely on another user who needs that console displacing him. If, in addition, a console is dedicated to a specific user, then it is not necessary to provide elaborate file protection since access to certain files can be limited to a given console input.

Another difficulty that should not be overlooked is the difficulty of using equipment being maintained by engineers from more than one other organization. If your computer is maintained by its manufacturer, then the problem of allocating the responsibility for an error between the computer equipment and the telephone equipment is yours. This can become especially difficult when dealing with a large monolithic organization such as the telephone company. An example of a problem that we encountered in this respect was that the telephone company was at first unwilling to tell us what equipment was available, but rather, wanted to know what our needs were, then they would tell us what we should use. We found this unacceptable for a research investigation.

D. Line Numbering

Two types of character consoles are available currently, the typewriter-like console and the character display C.R.T. The former is less costly, and does not have to be placed close to an expensive controller which services order of 8 displays. We are using the former, so the remarks will apply only to those consoles.

The problem to be tackled is one of modifying information in a file, stored, say, on the disk. In order to change any section, it is necessary to have some way of referring to it. To discuss the problem in concrete terms, let us assume that the file is divided into lines which are in a given order. There are three common ways that we could reference a particular line in the file--by its position (e.g. the 143rd), by content (e.g. the line that starts ABC), or by means of an additional ordered identifier, which we will call a "line number". Line numbers do not correspond to positions directly since the numbers need not be in an arithmetic progression. (e.g. 3, 10, 20, 25, 30 is a perfectly good ordered sequence of line numbers for referring to the (physically)

1st, 2nd, 3rd, 4th, and 5th lines in a file.)

A combination of physical position and content addressing could probably be used if a C.R.T. display with a fast enough display change were available since it would be possible to "drive through" the file, visually scanning for the desired line. A typewriter is extremely limited in output capability, so it can only be used for checking. We attempted to use a content and physical addressing scheme for file maintenance, but it proved to be unacceptable in use. Operations were provided to move a 'pointer' through the file, either by a given number of lines or until a given content was located (e.g. the word 'ABCDEF' in character positions 8 through 13). Lines could be deleted, printed, or inserted at the position of the pointer. The idea behind this was that most files are programs in some language, and that programs can be indexed by location symbols or statement numbers etc. Although, from a user point of view this could be as good as a line numbering scheme, there were a number of practical drawbacks, all to do with speed. We have switched to line numbering, so, in order to make a comparison, we will describe the line numbering scheme.

Each line has an associated number (in our case 3 digits, point, 3 digits). As lines of file are generated originally they are numbered, either by the user, or automatically by the system. It is usual to number them so that there are gaps between successive numbers, for example, 1.000, 2.000, etc. A line is inserted in a file by typing a number followed by the line. Lines are sorted in order of line number. Thus 1.500 is between 1 and 2. A line is deleted by typing its number and a blank line. In the implementation, the actual file is not changed each time that a change is typed, rather, an associated file of 'changes' is generated. Changing a file is costly because parts of it must be sorted and copied, usually involving time-consuming disk-head movement. Therefore, we only merge the changes into a file when it is to be used in an operation that requires it to be sorted (e.g. translation and listing).

By comparison, a content and physical location method requires that the file always be kept in order. If this is done by chaining, some disk manipulation is saved. On the other hand, searching, slow if the file is in order, is even slower if chaining is used. In a nutshell, our experiment with a non-line numbering scheme gave a negative result. Even if we were to improve it, we do not believe that it would be as useful as a line numbered scheme, so we have switched rather than fight.

E. Debugging Programs

One of the virtues of time-sharing schemes is that you are supposed to be able to debug programs more quickly at a console. For purposes of discussion, let us assume that the program is to be used for computations of a non-trivial nature at a later date, so that it must be written in an acceptable language such as Fortran, assembly language, Lisp, Algol, etc. Anyone can see that it is better to work at a console in such a language by trying the G.E./Dartmouth system, for example. Indeed, for programs up to a few hundred lines, fairly straightforward systems are very good. But what happens when a few thousand lines of code are being debugged?

If the source program is modified and retranslated, the amount of time is excessive (remember that compile time must be multiplied by the number of active users to estimate the delay). Because of a lack of facilities in our temporary system, we have found one of the best techniques for debugging using our temporary system to be octal patching. If a large program is being developed, we would modify it in octal from a console and check the changes, and then, when a sufficient number have been added, leave an updated source program for retranslation as a background program. What is needed in order to provide a way around this out-of-date debugging technique? The problem in the temporary system was that it was not convenient to partition a job into many subroutine segments and then only modify those of interest. (This is missing in the G.E./Dartmouth system which allows no segmentation.) The new system will encourage the user to keep his job broken down into many small subroutines. It will only be necessary for him to specify the name of the main program that he wishes to execute; the system will search his files for all subroutines that he calls. If a segment is to be changed, then the user types the file name of the segment and the changes. He must then ask for it to be translated. The relocatable binary object is kept with the source file until any further changes are made to that file. When the subroutine is to be loaded and executed, the loader fetches the binary version. It is planned that if there is no binary version, the source will be automatically translated. This organization ensures that a file is never translated more than once between changes.

When an interactive compiler compatible with Fortran is available, the user will be in a position to write and do most of the debugging of individual subroutines interactively, and then to store the programs as files in order

to test them as a group. Final changes can be made in source language to any one of the subroutines.

F. Special Purpose Software

The biggest single criticism of time sharing is that the overhead of system time is enormous. The usual answer to this is that the total cost of a job is reduced because less computer and man time is needed. This may be true, but it doesn't absolve us from the responsibility of finding more efficient methods.

Much of the overhead is due to the necessity of either having enough main memory and hardware to provide the flexible relocation so that many processes can be waiting to be executed, or having to frequently swap large amounts of information with disk file. Shared programs can reduce this considerably, but in order to make shared program effective, it is necessary to arrange that more than one user will be needing the same program nearly simultaneously. This means that the prospect of every user with his personally customized version of a compiler is not going to be efficient.

What must be done is to attempt to group users with like interests onto the same system and to provide a few flexible compilers or problem-oriented languages for their area. In other words, special-purpose programs are going to be more efficient, and, for that matter, better appreciated by users with those special needs.

One large use of time-sharing systems is for programs that are barely beyond the desk calculator stage. The Joss system was based on such problems; most other systems provide some equivalent. It has been rather sobering to note that the majority of users of the ILLIAC system use only the desk calculator program (not because of the absence of desk calculators)--in fact, the system is generally known by the name of that special-purpose program.

5. The Future System-Hardware

An IBM 360/50 is due for delivery 2Q 67 with a model 75 to follow shortly. The purpose of this system, in which the 50 will be a support processor for the 75, is to serve the campus computing needs, both in batch processing, low-speed terminals and special on-line processing for other departmental computers. Low-speed terminals will be connected via the PDP-7, so that it must be interfaced to the 50. In addition, service will almost certainly have to be provided to the Physics S.M.P. (Scanning and Measuring Projectors) via the CSX-1, to Civil Engineering who plan to implement a special purpose console system on a 360/40, to ILLIAC III, to ILLIAC IV, and to Chemistry who have an IBM 1800 on order for experimental instrumentation and control (see Figure 2). Therefore, these computers will be interfaced to the 50 in the future. A few additional small computers can be expected to require service in the lifetime of the system. In particular, a graphics terminal, the D.E.C.-338, will be connected via the P.D.P.-7.

6. Program Requirements

Regular batch processing should be allowed from any of several read stations, some of which may eventually be remote. Low-speed terminals should provide one or more languages of an interactive compiler type which will provide a capability extending from a simple desk calculator to a major subset of a standard language in the batch system so that smaller programs can be debugged on line, and simpler calculations can be completed. Interfaced computers must have the capability of having short bursts of calculation (5-10 secs.) completed without too long a delay. No provision is planned for interactive execution with a large program from a standard low-speed terminal. That would only be partially possible from a larger terminal such as the D.E.C.-338 which can request a burst of calculation.

Because of the size and complexity of OS/360, it is desirable that it be modified as little as possible. Hence it is necessary that we work with whatever is available and fast enough for our use. The A.S.P. (Attended Support Processor) system is planned for use on the coupled 50/75. This allows the 50 to route jobs to the 75. The 75 functions under the control of OS except that system I/O is routed to the 50. The 50 software allows several devices to build

files, can schedule these files for processing on the 75, and allows programs to be brought into the memory of the 50 from disk in order to process input files.

Since one use, the high energy physics S.M.P. work, accounts for up to 25% of the total computer load, it is worth investigating the characteristics of this use. 20% is of the batch processing type in which several reels of tape or their equivalent must be scanned. It is probably tape limited on the 75. The remaining 5% is on-line work direct from the CSX-1. If the required program (probably 150 K bytes or more) can be loaded rapidly, then about 3 to 5 seconds of compute will be needed about every 1.5 minutes.

7. A Proposal

The following proposal is based on the requirements and the restrictions discussed in the previous section. The need for frequent service to on-line computers by the model 75 dictates some form of time sharing, whereas previous experience suggests that the overhead will be high if arbitrary numbers of programs are allowed to execute in a time slotted manner, assuming that some of these programs approach the full core size (256K bytes for the model 50, 512 K bytes for the 75). In order to avoid the overhead of swapping, the number of programs in execution on the model 75 should not exceed the space available in core. Initial indications are that 360 object code is virtually unrelocatable after load time, so it will not be possible to dynamically relocate as requirements change. Therefore, code will have to be loaded for a predetermined area of memory. Since the on-line physics requirements and similar requirements demand that the code be already in an absolute form on a high-speed device such as the drum of the model 75, a known area of memory must be allocated to such jobs.

The proposal is to partition the 75 core into two equal regions. Both would be connected to the 50 for I/O (this requires a modification to A.S.P.). The 50 would schedule work for both partitions so that one will only receive short jobs of $\leq$ N secs. (N about 5 to 15 seconds, it could change during the day). The bursts of calculation required by remote computers would be scheduled in this partition. They would, in general, get most of their code in absolute from the drum or disk. The second partition would receive all other

jobs for the 75. This organization would mean that high priority short jobs such as arise from S.M.P. could be serviced within 2N seconds, and that very short batch jobs could jump the queue of longer jobs. Long jobs, such as the physics batch work, could be scheduled in order of arrival without overly delaying these short jobs, and there is some possibility for overlap of I/O and process between I/O limited jobs such as the physics work and short process jobs from remote computers.

Remote teletypes would be serviced interactively entirely at the PDP-7 and model 50 level to avoid interference with the process load. The facilities provided at the keyboard should include file handling, simple calculation, program testing, and submission of files to the 75 for processing on longer runs. This can be done largely within the ASP system by using its facility to load programs from the file for processing input files. The PDP-7 will buffer lines of characters and transmit them, with console numbers, to the 50. It will periodically analyse these and transmit any output back to the PDP-7. Within the file handling system it should provide facilities to update files equivalent to the facilities in the present ILLIAC II system. (See the Time Sharers' Manual.) An additional way of building a file should be provided by an interactive compiler. As lines with numbers are typed to it, they should be saved as a file. If the interactive compiler is a subset of the IBM Fortran IV or PL/I, the file can also be used within the batch processor at a subsequent time.

The desk calculator and the interactive compiler can conveniently be combined by using the approach of Iverson's APL (Arithmetic Programming Language). If an expression is typed, it is evaluated and printed. Thus if the user wants to know $\sqrt{3.1}$, he types

```
SQRT (3.1)
```

and immediately gets the answer. If he types

```
A = SQRT (3.1)
```

the effect is to define a variable A (if it does not already exist) and set it to $\sqrt{3.1}$. On the other hand, if a line of code is required, it is given a line number, for example

```
3.12    A = SQRT (3.1)
```

inputs a line of code for compilation but not execution.

8. Comments

The proposal is based partly on the assumption that dynamic relocation will not be practical. If it is, then additional partitions could be used, although the cost of moving programs and scheduling will increase. The important areas that need investigation are:

a. Is there sufficient file storage on the model 50 for the needs of remote uses and ASP?

b. Is there enough drum space for the 75 system and some of the high use processors for remote computers?

c. If not, can a dynamic use of the drum as in ILLIAC II be implemented?

d. Can sufficient overlap of I/O be achieved with only two partitions? (This perhaps should be simulated.)

e. What trade off between speed and size of an interactive compiler is best?

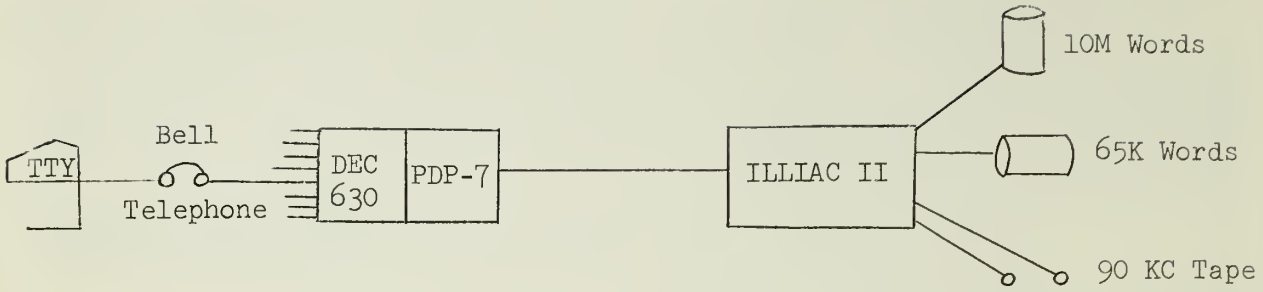

Figure 1

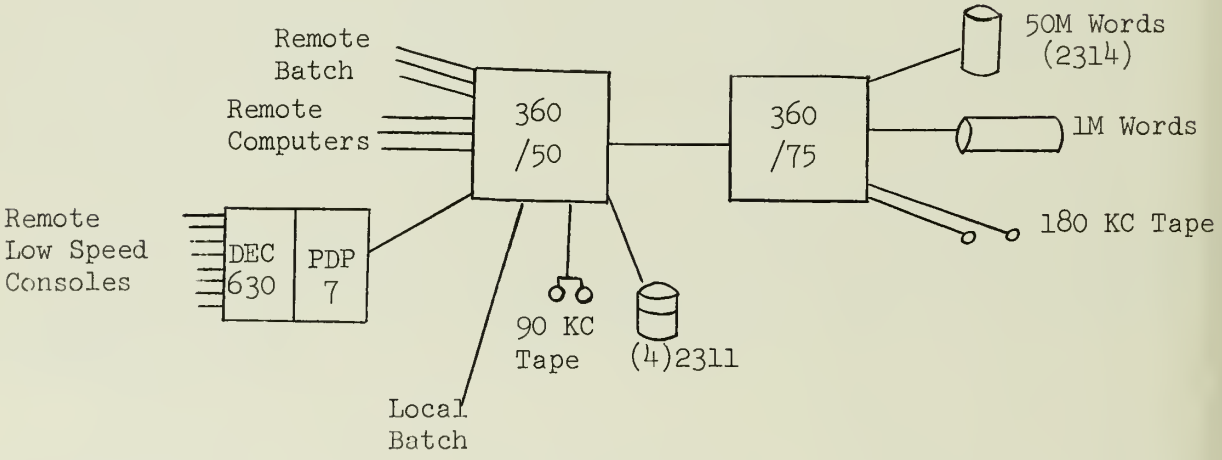

Figure 2